



Comandos de Decisão, Desvio e Repetição



COMANDO IF-THEN-ELSE

Comandos de controle do fluxo de execução permitem alterar o fluxo linear de execução de um programa. Há dois comandos de decisão: IF-THEN-ELSE e CASE.

O comando IF-THEN-ELSE executa diferentes trechos de programa em função do valor de uma condição (TRUE ou FALSE). Há três formas para o comando:

```
IF condição THEN
   executa_se_verdadeiro;
END IF;

IF condição THEN
   executa_se_verdadeiro;
ELSE
   executa_se_falso;
END IF;

IF condição1 THEN
   executa_se_verdadeiro1;
ELSIF condição2 THEN
   executa_se_verdadeiro2_e_falso1;
[ELSIF condição3 THEN]
   [executa_se_verdadeiro3_e_falso1_e_falso2;] ...
[ELSE]
   [executa_se_falso1_e_falso2_e_falso3_e_ ...;]
END IF;
```

Na primeira forma, condição é testada e um bloco de comandos é executado se condição = TRUE. O programa continua no comando imediatamente após a cláusula END IF.

Na segunda, condição é testada e um primeiro bloco de comandos é executado se condição = TRUE e um segundo bloco, se condição = FALSE (cláusula ELSE). Observe que apenas um dos blocos é executado, já que a condição = TRUE ou condição = FALSE, mas não ambos. Após a execução do bloco correspondente, o programa continua no comando imediatamente após a cláusula END IF.

A terceira forma permite testar diversas condições. Se a condição = TRUE, o bloco correspondente é executado. Caso contrário, a condição seguinte (cláusula ELSIF) é testada e assim por diante. Se nenhuma das condições for TRUE, o bloco de comandos da cláusula ELSE, se esta estiver presente, é executado. Apenas um dos blocos de comandos será executado. Após a execução do bloco correspondente, o programa continua no comando imediatamente após a cláusula END IF.

A cláusula ELSIF condição, embora seja equivalente a ELSE IF condição, é de mais fácil leitura e evita que sejam utilizados IF aninhados.

O exemplo a seguir ilustra o funcionamento do comando IF.



```
DECLARE
    a NUMBER;
    b NUMBER;
BEGIN
    a := 0; b := 0;
    IF a = b THEN
        DBMS_OUTPUT.PUT_LINE('a igual a b');
    ELSIF a > b THEN
        DBMS_OUTPUT.PUT_LINE('a maior que b');
    ELSIF a < b THEN
        DBMS_OUTPUT.PUT_LINE('a menor que b');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Oooooops!');
    END IF;
END;
```

O resultado, após a execução do bloco acima, é:

```
a igual a b
```

Se este mesmo bloco fosse executado com *a* := NULL ou *b* := NULL, o resultado seria:

```
'Oooooops!'
```

Isto ocorre porque na maioria, se não em todos os SGBDR, NULL ≠ NULL. Pode parecer estranho, mas o valor NULL pode ter significados diferentes para situações distintas: indefinido, inexistente ou não aplicável. Como o Oracle não sabe o que cada NULL representa, optou-se por definir este padrão. Na realidade, não se deve comparar variáveis ou expressões com valor NULL.

Pode-se testar se uma variável ou expressão é NULL utilizando a condição pré-definida IS NULL ou sua forma negativa IS NOT NULL:

```
IF x IS [NOT] NULL THEN ...
```

Alterando-se o exemplo anterior para tratar valores NULL, tem-se:

```
DECLARE
    a NUMBER;
    b NUMBER;
BEGIN
    IF a = b THEN
        DBMS_OUTPUT.PUT_LINE('a igual a b');
    ELSIF a > b THEN
        DBMS_OUTPUT.PUT_LINE('a maior que b');
    ELSIF a < b THEN
        DBMS_OUTPUT.PUT_LINE('a menor que b');
    ELSIF (a IS NULL) OR (b IS NULL) THEN
        DBMS_OUTPUT.PUT_LINE('Pelo menos um dos valores é NULL');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Oooooops!');
    END IF;
END;
```

Agora, executando o bloco de comandos com a ou b = NULL, o resultado é corretamente apresentado. As cláusulas ELSE e ELSIF são opcionais. Porém, se forem utilizadas, elas devem conter pelo menos um comando válido.

COMANDO CASE

O comando CASE é similar ao comando IF: permite escolher um bloco de comandos para ser executado dentre vários possíveis. Há duas formas para o comando CASE:



```
CASE expressão
WHEN valor1 THEN
        bloco_de_comandos1;
[WHEN valor2 THEN
        bloco_de_comandos2;]
...
[ELSE
        bloco_de_comandos_else;]
END CASE;
```

```
CASE
WHEN condição1 THEN
        bloco_de_comandos1;
[WHEN condição2 THEN
        bloco_de_comandos2;]
...
[ELSE
        bloco_de_comandos_else;]
END CASE;
```

Na primeira forma expressão é avaliada. Em seguida, o valor calculado é comparado com valor1. Se forem iguais, bloco_de_comandos1 é executado e o programa continua após a cláusula END CASE. Se não forem iguais, o valor

de expressão é comparado com valor2 e assim por diante. Se nenhum dos valores fornecidos for igual ao valor de expressão, o bloco de comandos da cláusula ELSE é executado. Observe que penas um dos blocos de comando do CASE é executado.

Embora a cláusula ELSE seja opcional, um erro ocorre se ela for omitida e nenhum dos valores definidos fornecidos for igual ao valor de expressão. Em resumo, apesar de opcional, a cláusula ELSE deve ser utilizada para que não ocorram erros.

Na segunda forma, as condições definidas nas cláusulas WHEN são avaliadas. O bloco de comandos correspondente à primeira condição verdadeira é executado. Caso nenhuma das condições seja verdadeira, o bloco de comandos da cláusula ELSE é executado. O mesmo se aplica à cláusula ELSE.

Sempre é possível converter um bloco IF-THEN-ELSIF-ELSE em um bloco correspondente CASE-WHEN-ELSE na segunda forma e vice-versa. O exemplo anterior é reescrito utilizando-se o comando CASE.

```
DECLARE
    a NUMBER;
    b NUMBER;
BEGIN
    a := NULL; b := 0;
    CASE
        WHEN a = b THEN
            DBMS_OUTPUT.PUT_LINE('a igual a b');
        WHEN a > b THEN
            DBMS_OUTPUT.PUT_LINE('a maior que b');
        WHEN a < b THEN
            DBMS_OUTPUT.PUT_LINE('a menor que b');
        WHEN (a IS NULL OR b IS NULL) THEN
            DBMS_OUTPUT.PUT_LINE('Pelo menos um dos valores é NULL');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Oooooops!');
    END CASE;
END;
```

O resultado é idêntico ao do exemplo anterior.

Não há regra geral sobre quando usar o comando IF ou o CASE. Pode-se dizer que, quando se deseja testar valores diferentes para uma mesma variável ou expressão, a primeira forma do CASE torna o código mais legível. Já nas situações envolvendo condições mais complexas, envolvendo

variáveis diferentes, tanto faz o IF ou a segunda forma do CASE.

Um comando que pode ser muito útil durante a fase de desenvolvime de uma aplicação é o comando NULL. Ele simplesmente não faz coisa alguma. Pode-se utiliza-lo como comando obrigatório das cláusulas ELSE e ELSIF do comando IF ou ELSE do comando CASE. O trecho de programa a seguir parece não fazer coisa alguma (e não faz mesmo!), mas pode ser apenas a estrutura de um bloco de programa, que será preenchido mais tarde com os comandos necessários:

```
IF (salario > 10000) AND (regime_trabalho = 'CLT') THEN
   NULL;
   -- colocar aqui o cálculo do IRPF ...
ELSIF (regime_trabalho = 'RPA') then
   NULL;
   -- colocar aqui o cálculo do dos descontos de IRPF e ISS ...
.
.
.
```

É comum, durante o início da fase de desenvolvimento, se preocupar mais com a estrutura do programa. Para isto, o comando NULL é bastante útil.

CASE EM EXPRESSÕES (EXPRESSION CASE)

Os comandos IF e CASE são utilizados para controlar a execução de blocos de comando. Eles não podem ser utilizados diretamente em expressões. Há, porém, uma forma do comando CASE que pode ser utilizado em expressões:



```
CASE expressão
WHEN valor1 THEN
     valor_retorno1
[WHEN valor2 THEN
     valor_retorno2]
...
[ELSE
     valor_retorno_else]
END
```

```
CASE
WHEN condição1 THEN
     valor_retorno1
[WHEN condição2 THEN
     valor_retorno2]
...
[ELSE
     valor_retorno_else]
END
```

As diferenças são pequenas, porém importantes: (i) ao invés de um bloco de comandos, cada cláusula WHEN deve ter um valor ou expressão, (ii) não deve ser colocado o ponto-e-vírgula entre as cláusulas WHEN e (ii) o comando é terminado por END apenas. No exemplo a seguir, o nome do dia da semana é impresso em função do valor da variável dia (1 – segunda-feira e 7 – domingo).

```
DECLARE
    dia NUMBER;
    nome VARCHAR2(15);
BEGIN
    -- Exibe o nome do dia da semana (1 - segunda, 7 - domingo)
    dia := 5;
    nome := CASE dia
                WHEN 1 THEN 'segunda'
                WHEN 2 THEN 'terca'
                WHEN 3 THEN 'quarta'
                WHEN 4 THEN 'quinta'
                WHEN 5 THEN 'sexta'
                WHEN 6 THEN 'sábado'
                WHEN 7 THEN 'domingo'
                ELSE '**erro**'
            END ||
            CASE
                WHEN dia IN (1, 2, 3, 4, 5) THEN '-feira'
                ELSE ''
            END;
    DBMS_OUTPUT.PUT_LINE(nome);
END;
```

O resultado após a execução do bloco anônimo é:

```
sexta-feira
```

CASE pode ser utilizado em qualquer expressão, até mesmo em consultas SQL. Deve-se tomar cuidado, no entanto, com os tipos dos valores retornados.

COMANDO GOTO

O comando GOTO desvia a execução do programa para o comando seguinte ao rótulo indicado. Sua forma geral é:

GOTO rótulo;



Algumas restrições se aplicam ao comando GOTO:

- Não é permitido transferir a execução para dentro de comandos IF, CASE e LOOP;
- Não é permitido transferir a execução para dentro de um sub-bloco;
- Não é permitido transferir a execução de uma cláusula para outra de comandos IF e CASE. Por exemplo, não é permitido utilizar o comando GOTO para transferir a execução da cláusula ELSIF para dentro da cláusula ELSE de um comando IF;
- Não é permitido transferir a execução de um bloco para a seção de tratamento de exceções e vice-versa.

Os exemplos mostrados a seguir não são permitidos:

```
...
BEGIN
    GOTO goto_dentro_de_if;
    IF x = 0 THEN
        <<goto_dentro_de_if>>
        x := x + 1;
    ...
    END IF;
...
END;

...
BEGIN
    DECLARE
        y INT;
    BEGIN
        <<goto_dentro_de_sub-bloco>>
        ...
    END;
    GOTO goto_dentro_de_sub-bloco;
    ...
END;
```

```
...
BEGIN
    CASE x
        WHEN 0 THEN
            <<goto_dentro_de_case>>
            x := x + 1;
        WHEN 1 THEN
            GOTO <<goto_dentro_de_case>>;
        ...
    END CASE;
...
END;
```

Deve-se ter muito cuidado ao se utilizar o comando GOTO. É muito fácil escrever programas com fluxo de execução muito complexo e difícil de se acompanhar, os chamados código espaguete, comuns na época em que linguagens como BASIC e COBOL eram largamente utilizadas. No exemplo a seguir, é implementado o cálculo do fatorial de um número inteiro (n! = n·(n – 1)·(n – 2) ... 2·1) utilizando-se o comando GOTO.

```
DECLARE
    fatorial NUMBER := 1;
    n NUMBER;
    i NUMBER := 0;
BEGIN
    n := 6;
    i := n;
    <<inicio_loop>>
    IF i >= 1 THEN
        fatorial := fatorial * i;
        i := i - 1;
        GOTO inicio_loop;
    END IF;
    DBMS_OUTPUT.PUT_LINE('O fatorial de ' || n || ' é ' || fatorial);
END;
```

O resultado após a execução do bloco acima é:

```
O fatorial de 6 é 720
```

COMANDO LOOP

Comandos de repetição controlam a execução repetitiva de blocos de comando, também chamada de loops (laços, em português). São 3 os comandos de repetição em PL/SQL: LOOP, FOR e WHILE. Juntamente com eles, utilizam-se também os comandos EXIT e CONTINUE.

Na teoria, qualquer um dos 3 comandos pode ser utilizado para implementar

qualquer loop. Na prática, porém, a escolha de que comando usar depende das características do loop.

O primeiro comando, LOOP, implementa o chamado loop simples. Sua forma geral é mostrada a seguir:

```
[<<rótulo_loop>>;]
LOOP
    bloco_de_comandos;
    [EXIT [rótulo] [WHEN condição];]
    [CONTINUE [rótulo] [WHEN condição];]
    [GOTO rótulo;]
END LOOP [<<rótulo_loop>>];
```

O bloco de comandos é repetido indefinidamente até que seja encontrado um comando EXIT (saída incondicional do loop), EXIT WHEN (saída do loop se condição = TRUE) ou GOTO. Todos os três podem aparecer repetidamente e em qualquer parte do bloco de comandos. É muito importante que a condição de saída seja incluída, sob o risco de o bloco ficar indefinidamente em execução (loop infinito).

Os comandos EXIT e EXIT WHEN, quando utilizados sem um rótulo, transferem a execução do programa para o comando imediatamente após o fim do loop corrente. Quando utilizados com rótulo, transferem a execução do programa para o comando imediatamente após o fim loop identificado pelo rótulo. Neste último caso, o rótulo utilizado deve necessariamente rotular um loop (deve ser declarado imediatamente antes do comando LOOP) e este loop rotulado deve conter o loop onde está o comando EXIT. Veja os exemplos a seguir:

(a)

```
1  DECLARE
2  ...
3  BEGIN
4    <<r1>>
5    LOOP
6      <<r2>>
7      EXIT WHEN condição1;
8      ...
9      <<r3>>
10     LOOP
11       EXIT r1 WHEN condição2;
12       ...
13       <<r4>>
14       LOOP
15         EXIT WHEN condição3;
16         ...
17       END LOOP r3;
18     END LOOP r2;
19   END LOOP r1;
20   <<r5>>
21   LOOP
22     EXIT WHEN condição4
23     ...
24   END LOOP r5;
25 END;
```

(b)

```
1  DECLARE
2  ...
3  BEGIN
4    <<r1>>
5    LOOP
6      <<r2>>
7      EXIT WHEN condição1;
8      ...
9      <<r3>>
10     LOOP
11       EXIT r2 WHEN condição2;
12       ...
13       <<r4>>
14       LOOP
15         EXIT WHEN condição3;
16         ...
17       END LOOP r3;
18     END LOOP r2;
19   END LOOP r1;
20   <<r5>>
21   LOOP
22     EXIT WHEN condição4
23     ...
24   END LOOP r5;
25 END;
```

(c)

```
1  DECLARE
2  ...
3  BEGIN
4    <<r1>>
5    LOOP
6      <<r2>>
7      EXIT WHEN condição1;
8      ...
9      <<r3>>
10     LOOP
11       EXIT r5 WHEN condição2;
12       ...
13       <<r4>>
14       LOOP
15         EXIT WHEN condição3;
16         ...
17       END LOOP r3;
18     END LOOP r2;
19   END LOOP r1;
20   <<r5>>
21   LOOP
22     EXIT WHEN condição4
23     ...
24   END LOOP r5;
25 END;
```

O exemplo mostrado em (a) compilará corretamente. O comando EXIT, na linha 11, faz referência ao rótulo do loop mais externo. O exemplo em (b) não compilará, pois o comando EXIT faz referência a um rótulo que não rotula um loop. O exemplo em © também não compilará, pois o comando EXIT faz referência a um rótulo em um loop que não o contém.

Os comandos CONTINUE e CONTINUE WHEN interrompem incondicional ou condicionalmente a execução da iteração atual, transferindo a execução para o primeiro comando após o LOOP ao qual pertencem. As mesmas restrições vistas no comando EXIT se aplicam ao CONTINUE, caso este faça referência a um rótulo.

Já o comando GOTO não tem estas restrições. O GOTO transfere a execução para o comando imediatamente após rótulo especificado, obedecidas as restrições naturais do GOTO (desvio para dentro de IF, CASE, sub-bloco etc.).

O exemplo a seguir apresenta duas formas do fatorial de um número inteiro utilizando o comando LOOP:

```
DECLARE
    fatorial NUMBER := 1;
    n NUMBER;
    i NUMBER;
BEGIN
    n := 6;
    i := n;
    LOOP
        IF i <= 1 THEN
           EXIT;
        END IF;
        fatorial := fatorial * i;
        i := i - 1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('O fatorial de ' || n
        || ' é ' || fatorial);
END;
```

```
DECLARE
    fatorial NUMBER := 1;
    n NUMBER;
    i NUMBER;
BEGIN
    n := 1;
    i := n;
    LOOP
       EXIT WHEN i <= 1;
       fatorial := fatorial * i;
       i := i - 1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('O fatorial de ' || n
        || ' é ' || fatorial);
END;
```

O resultado, após a execução, é:

```
6! = 720
```

O primeiro programa utiliza o comando EXIT enquanto o segundo utiliza EXIT WHEN. A segunda forma evita o uso do IF para testar a condição de saída. Agora, suponha que se deseje calcular o fatorial dos números pares entre dois números fornecidos. O programa a seguir implementa a lógica necessária.

```
DECLARE
    lim_inf NUMBER;
    lim_sup NUMBER;
    fatorial NUMBER;
    i NUMBER;
BEGIN
    lim_inf := 0; lim_sup := 12;
    -- Calcula os fatorial dos números pares >= lim_inf <= lim_sup.
    -- lim_inf é utilizado como contador. É necessário decrementa-lo
    -- porque a primeira coisa que é feita no loop é incrementá-lo.
    lim_inf := lim_inf - 1 ;
    <<loop_principal>>
    LOOP
        lim_inf := lim_inf + 1;
        -- Condição de saída do loop
        EXIT WHEN lim_inf > lim_sup;
        -- Volta para o início do loop se não for número par
        CONTINUE loop_principal WHEN MOD(lim_inf, 2) <> 0;
        -- É número par. Calcula o fatorial
        fatorial := 1;
        i := lim_inf;
        LOOP
            EXIT WHEN i <= 1;
            fatorial := fatorial * i;
            i := i - 1;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('O fatorial de ' || lim_inf || ' é '
            || TO_CHAR(fatorial, '999G999G999G999G999'));
    END LOOP;
END;
```

No exemplo acima, o comando EXIT WHEN é utilizado para encerrar o loop e o comando CONTINUE WHEN <rótulo> é utilizado para pular os núm ímpares. O rótulo utilizado para indicar o loop não é necessário, pois o CONTINUE faz referência ao próprio loop onde é executado. Uma novidade é a função TO_CHAR, que converte um valor numérico em texto e aceita um parâmetro com o formato a ser utilizado (pesquise a respeito desta função; você deverá utiliza-la com frequência!). O resultado, após a execução do bloco acima, é mostrado a seguir.

```
O fatorial de 0 é                1
O fatorial de 2 é                2
O fatorial de 4 é               24
O fatorial de 6 é              720
O fatorial de 8 é           40.320
O fatorial de 10 é       3.628.800
O fatorial de 12 é     479.001.600
```

Há diversas outras formas de se implementar a mesma lógica. Por exemplo, os valores de lim_inf e lim_sup podem ser ajustados para o número par igual ou superior (lim_inf) e igual ou inferior (lim_sup), antes do início do loop externo. Com isto, pode-se incrementar lim_inf de 2 a cada iteração. Esta implementação encontra-se a seguir.

```
DECLARE
    lim_inf NUMBER;
    lim_sup NUMBER;
    fatorial NUMBER;
    i NUMBER;
BEGIN
    lim_inf := 1; lim_sup := 11;
    -- Calcula os fatorial dos números pares >= lim_inf <= lim_sup.
    -- lim_inf é utilizado como contador.
    lim_inf := CASE MOD(lim_inf, 2) WHEN 1 THEN lim_inf + 1 ELSE lim_inf END;
    lim_sup := CASE MOD(lim_sup, 2) WHEN 1 THEN lim_sup - 1 ELSE lim_sup END;
        LOOP
        -- Condição de saída do loop
        EXIT WHEN lim_inf > lim_sup;
        fatorial := 1;
        i := lim_inf;
        LOOP
            EXIT WHEN i <= 1;
            fatorial := fatorial * i;
            i := i - 1;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE('O fatorial de ' || lim_inf || ' é '
            || TO_CHAR(fatorial, '999G999G999G999G999'));
        lim_inf := lim_inf + 2;
    END LOOP;
END;
```

O resultado, após a execução do bloco acima, é mostrado a seguir.

```
O fatorial de 2 é            2
O fatorial de 4 é           24
O fatorial de 6 é          720
O fatorial de 8 é       40.320
O fatorial de 10 é   3.628.800
```

COMANDO FOR

O comando FOR executa um bloco de comandos um número pré-definido de vezes. Sua forma geral é:

```
[<<rótulo_for>>;]
FOR índice IN [REVERSE] limite_inferior .. limite_superior LOOP
    bloco_de_comandos;
    [EXIT [rótulo] [WHEN condição];]
    [CONTINUE [rótulo] [WHEN condição];]
    [GOTO rótulo;]
END LOOP [<<rótulo_for>>];
```

Índice é a variável de controle do FOR. Ela é definida automaticamente e seu escopo é o corpo do FOR. Isto significa que ela é criada assim que o l começa, pode ser referenciada dentro do bloco de comandos do FOR, porém deixa de existir após seu fim. Índice não deve ser definido no bloco onde o FOR estiver.

Inicialmente, índice recebe limeite_inferior. Quando a execução chega ao END LOOP, índice incrementado de uma unidade. A execução do bloco de comandos termina quando índice for maior que limite_superior. Se limite_superior for maior que limite_inferior, o bloco de comandos do FOR não é executado e a execução continua após o END LOOP.

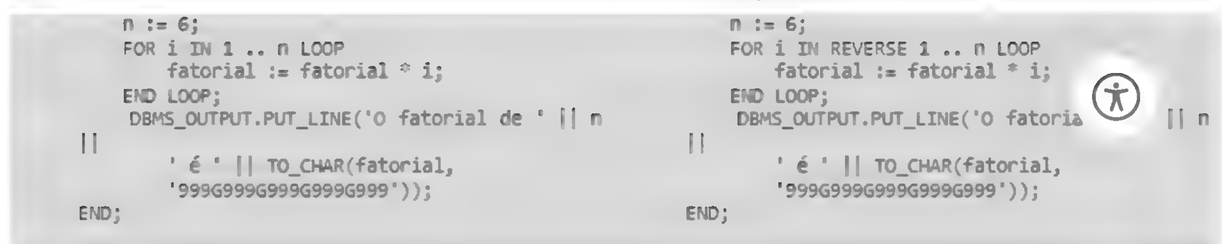
Tanto limite_inferior quanto limite_superior podem ser variáveis, literais ou expressões que retornem um valor numérico. Seus valores são arredondados automaticamente para o inteiro mais próximo. Se a cláusula REVERSE for utilizada, índice recebe inicialmente limeite_superior e seu valor é decrementado de uma unidade até que seja menor que limite_inferior.

Embora seja possível alterar os valores do índice e de limite_inferior e limite_supeiror, dentro do bloco de comandos, se forem utilizadas variáveis para estes dois últimos, a alteração não surtirá qualquer efeito. Os respectivos valores são copiados para variáveis internas e estas variáveis internas são utilizadas para controlar a execução do FOR.

Os comandos EXIT, EXIT WHEN, CONTINUE, CONTINUE WHEN e GOTO funcionam da mesma forma que no comando LOOP simples. Nos exemplos a seguir, a função fatorial é escrita utilizando-se as duas formas do comando FOR.

```
DECLARE
    fatorial NUMBER := 1;
    n NUMBER;
BEGIN
```

```
DECLARE
    fatorial NUMBER := 1;
    n NUMBER;
BEGIN
```



```
    n := 6;
    FOR i IN 1 .. n LOOP
        fatorial := fatorial * i;
    END LOOP;
     DBMS_OUTPUT.PUT_LINE('O fatorial de ' || n
||
        ' é ' || TO_CHAR(fatorial,
        '999G999G999G999G999'));
END;
```

```
    n := 6;
    FOR i IN REVERSE 1 .. n LOOP
        fatorial := fatorial * i;
    END LOOP;
     DBMS_OUTPUT.PUT_LINE('O fatorial de ' || n
||
        ' é ' || TO_CHAR(fatorial,
        '999G999G999G999G999'));
END;
```

O resultado exibido é o mesmo que o do exemplo anterior.

Como o comando FOR possui uma condição de saída implícita, não é necessária a presença dos comandos EXIT, CONTINUE ou GOTO para sair do bloco de comandos, embora seja possível utilizá-los. Observe que o índice i é referenciado normalmente dentro do bloco de comandos do FOR.

Referências a i fora do bloco de comandos causará um erro de compilação.

Em situações onde há um FOR LOOP dentro de outro FOR LOOP (FOR aninhados), é possível referenciar o índice do FOR LOOP externo dentro do bloco de comandos do FOR LOOP interno, mesmo que ambos os índices tenham o mesmo nome. Neste caso é necessário acrescentar um rótulo ao FOR LOOP. Observe sua utilização no exemplo a seguir, que implementa o cálculo do fatorial para uma faixa de números pares entre dois limites com o comando FOR LOOP.

Para melhorar a clareza e entendimento dos programas desenvolvidos, recomenda-se a utilização de nomes distintos e autoexplicativos para os índices tanto de FOR LOOP individuais quanto aninhados.

```
DECLARE
    lim_inf NUMBER;
    lim_sup NUMBER;
    fatorial NUMBER;
    n NUMBER;
BEGIN
    -- Calcula os fatorial dos números pares >= lim_inf <= lim_sup usando FOR.
    lim_inf := 0; lim_sup := 13;
    <<loop_externo>>
    FOR i IN lim_inf .. lim_sup LOOP
        CONTINUE WHEN MOD(i, 2) <> 0;
        fatorial := 1;
        -- Só é necessário qualificar índices de loops externos e caso haja
        -- coincidência de nomes.
        FOR i IN 2 .. loop_externo.i LOOP
            fatorial := fatorial * i;
        END LOOP;
        -- Melhorando a exibição do resultado
        DBMS_OUTPUT.PUT_LINE('O fatorial de ' || TO_CHAR(i, '99') || ' é '
            || TO_CHAR(fatorial, '999G999G999G999G999'));
    END LOOP;
END;
```

O resultado após a execução é mostrado a seguir.

```
O fatorial de  0 é                   1
O fatorial de  2 é                   2
O fatorial de  4 é                  24
O fatorial de  6 é                 720
O fatorial de  8 é              40.320
O fatorial de 10 é           3.628.800
O fatorial de 12 é         479.001.600
```

A maioria das linguagens de programação que implementam o comando FOR permitem definir o valor do incremento. Em PL/SQL, o incremento é sempre igual a 1 (ou –1, no caso de REVERSE). Os exemplos a seguir mostram como gerar a sequência 1, 3, 5, 7, 9 em Python e em Java:

```
Python

for x in range(1, 10, 2):
  print(i)
```

```
Java

for (int i = 1; i <= 9; i = i + 2) {
      System.out.println(i);
   }
```

Pode-se simular incrementos diferentes utilizando-se a função MOD (resto da divisão inteira). No último exemplo apresentado, o comando CONTINUE WHEN MOD(i, 2) <> 0, colocado imediatamente após o FOR LOOP, emula um

incremento igual a 2. De forma geral, para emular um incremento igual a n, deve-se utilizar CONTINUE WHEN MOD(i, n) = MOD(limite_inferior, n). A solução acima funciona apenas para limites inferior, superior e incremento positivos. Veja a sua utilização no exemplo a seguir.

```
DECLARE
    lim_inf NUMBER;
    lim_sup NUMBER;
    incremento NUMBER;
BEGIN
    -- Emulando incrementos diferentes de 1 para o comando FOR
    lim_inf := 5; lim_sup := 34; incremento := 4;
    FOR i IN lim_inf .. lim_sup LOOP
        CONTINUE WHEN MOD(i, incremento) <> MOD(lim_inf, incremento);
        DBMS_OUTPUT.PUT_LINE(i);
    END LOOP;
END;
```

O resultado após a execução é mostrado a seguir.

```
5
9
13
17
21
25
29
33
```

COMANDO WHILE

O comando WHILE executa um bloco de comandos enquanto uma determinada condição for verdadeira. Sua forma geral é:

```
[<<rótulo_while>>:]
WHILE condição
```

```
    bloco_de_comandos;
    [EXIT [rótulo] [WHEN condição];]
    [CONTINUE [rótulo] [WHEN condição];]
    [GOTO rótulo;]
END LOOP [<<rótulo_while>>];
```

O bloco de comandos é executado enquanto condição for verdadeira. Se condição for inicialmente falsa, o programa continua a partir do comando imediatamente após o END LOOP. No exemplo a seguir, a função fatorial é implementada utilizando-se o comando WHILE.

```
DECLARE
    n NUMBER;
    i NUMBER;
    fatorial NUMBER := 1;
BEGIN
    n := 6;
    i := n;
    WHILE i > 1 LOOP
        fatorial := fatorial * i;
        i := i - 1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('O fatorial de ' || TO_CHAR(n, '99') || ' é '
        || TO_CHAR(fatorial, '999G999G999G999G999'));
END;
```

O resultado após a execução é mostrado a seguir.

```
O fatorial de  6 é              720
```

Pode-se utilizar o comando WHILE para emular o comando FOR com incrementos diferentes de 1 e –1. Deve-se tomar cuidado apenas com a atribuição de valores a lim_inf e lim_sup para que se possa emular corretamente o comando FOR com e sem a cláusula REVERSE. Veja o exemplo a seguir.

```
DECLARE
    lim_inf NUMBER;
    lim_sup NUMBER;
    incremento NUMBER;
    ordem BOOLEAN;
BEGIN
    -- Comando WHILE emulando o comando FOR com (ordem = TRUE) e sem (ordem = F)
    -- a cláusula REVERSE e com qualquer valor de incremento (incrementos para o
    -- FOR com REVERSE devem ser negativos).
    lim_inf := -1; lim_sup := -35; incremento := -4; ordem := TRUE;
    WHILE TRUE LOOP
        EXIT WHEN CASE ordem WHEN TRUE THEN lim_inf < lim_sup ELSE lim_inf > lim_sup END;
        -- lim_inf corresponde ao índice do FOR.
        DBMS_OUTPUT.PUT_LINE(lim_inf);
        lim_inf := lim_inf + incremento;
    END LOOP;
END;
```

A seguir, os resultados apresentados após a execução, para diversos valores de lim_inf, lim_sup, incr e ordem.

lim_inf := 5; lin_sup := 34; incr := 4; ordem := FALSE;	lim_inf := 5; lin_sup := 34; incr:= 4; ordem := TRUE;	lim_inf := 34; lin_sup := 5; incr:= -4; ordem := TRUE;	lim_inf := -5; lin_sup := 34; incr:= 4; ordem := FALSE;	lim_inf := -35; lin_sup := -1; incr:= 4; ordem := FALSE;	lim_inf := -1; lin_sup := -35; incr:= -4; ordem := TRUE;
5 9 13 17 21 25 29 33		34 30 26 22 18 14 10 6	-5 -1 3 7 11 15 19 23 27 31	-35 -31 -27 -23 -19 -15 -11 -7 -3	-1 -5 -9 -13 -17 -21 -25 -29 -33

O comando WHILE testa a condição antes de cada execução do bloco de comandos. Em algumas situações, deseja-se que o bloco de comandos seja executado pelo menos uma vez e que a condição seja testada ao final. Algumas linguagens implementam o comando DO bloco_de_comados UNTIL condição, onde o bloco de comandos é executado até que condição seja verdadeira.

```
Bloco LOOP ... UNTIL condição

LOOP
  bloco_de_comandos;
UNTIL condição;
```

```
Trecho equivalente utilizando WHILE

WHILE TRUE LOOP
  bloco_de_comandos;
  EXIT WHEN condição;
END LOOP;
```

As mesmas restrições aos comandos EXIT, CONTINUE e GOTO se aplicam ao WHILE.

A decisão de que comando utilizar depende da situação. Como regra geral, utilizam-se os comandos LOOP e WHILE quando não se sabe quantas vezes o bloco de comandos deverá ser executado, sendo que, para o LOOP, o bloco de comandos será executado pelo menos uma vez. Já o comando FOR deve ser utilizado quando se deseja que o bloco de comandos seja executado um determinado número de vezes.
Embora os comandos de repetição, juntamente com os comandos EXIT, CONTINUE e GOTO ofereçam uma enorme flexibilidade, eles também podem se tornar uma grande dor de cabeça se algumas diretrizes não forem observadas. São elas:

- Utilize variáveis (e índices) com nomes intuitivos. Vários exemplos apresentados ao longo do texto utilizam variáveis com nomes genéricos (i, j, k etc.). No entanto, são trechos muito pequenos de programa que cabem facilmente em uma única tela de um editor de textos. Lembre-se que você ou outras pessoas deverão manter o programa no futuro. Não faz diferença, em termos de desempenho, o tamanho do nome da variável. Todas as referências às variáveis declaradas se tornam endereços de memória após a compilação;

- Existe uma máxima em programação estruturada que diz "uma única entrada e uma única saída". Isto significa que os programas (e suas estruturas de controle) devem ter pontos únicos de entrada e saída. Para os comandos de repetição, há sempre uma única entrada, mas pode haver muitos pontos de saída. Evite isto sempre que possível;

- Evite usar o comando GOTO. É muito fácil perder o controle do programa com GOTO;

- Não utilize os comandos EXIT, EXIT WHEN, CONTINUE e CONTINUE WHEN com o comando FOR. O comando FOR existe para que o bloco de comando seja repetido um número determinado de vezes;
  O mesmo se aplica ao comando WHILE. Inclua na condição a lógica que seria utilizada com o EXIT;

- Utilize rótulos para nomear comandos de repetição e seus respectivos END LOOP, principalmente em programas grandes, com muitos comandos aninhados. É muito ruim ter que acompanhar o início e fim de blocos de comandos, principalmente se você tiver que rolar a tela várias vezes;

- Coloque um comentário após o END IF com a condição testada no respectivo IF;

- Comente o programa, principalmente aqueles que implementam algoritmos geniais, porém pouco intuitivos.

Referência Bibliográfica

- FEUERSTEIN, S. **Oracle PL/SQL Programming**. 6a Ed., O'Reilly, 2014.

- PUGA, S., FRANÇA, E. e GOYA, M. **Banco de Dados: Implementação em SQL, PL SQL e Oracle 11g**. São Paulo: Pearson, 2014.



- Gonçalves, E. **PL/SQL: Domine a linguagem do banco de dados Oracle**. Versão Digital. Casa do Código, 2015.

- GROFF, J. R., WEINBERG, P. N. e OPPEL, A. J. **SQL: The Complete Reference**. 3a Ed., Nova York: McGraw-Hill, 2009.

- ELMASRI, R. e NAVATHE, S. B. **Sistemas de Banco de Dados**. 7ª Ed., São Paulo: Pearson, 2011.

Ir para exercício